4) Post-IFFT complex multiply step:

Compute N/8-point complex multiplication products y1[n] and y2[n], n=0,1,...N/8-1.

Pseudo code

```
for(n=0; n<N/8; n++)
{
    /* y1[n] = z1[n] * (xcos2[n] + j * xsin2[n]) ; */
    y1[n] = (zr1[n] * xcos2[n] - zi1[n] * xsin2[n]) + j * (zi1[n] * xcos2[n] + zr1[n] * xsin2[n]) ;
    /* y2[n] = z2[n] * (xcos2[n] + j * xsin2[n]) ; */
    y2[n] = (zr2[n] * xcos2[n] - zi2[n] * xsin2[n]) + j * (zi2[n] * xcos2[n] + zr2[n] * xsin2[n]) ;
}
```

where

zr1[n] = real(z1[n]) ;
zi1[n] = imag(z1[n]) ;
zr2[n] = real(z2[n]) ;
zi2[n] = imag(z2[n]) ;

and xcos2[n] and xsin2[n] are as defined in step 2 above.

5) Windowing and de-interleaving step.

Compute windowed time-domain samples x[n].

Pseudo code

```
for(n=0; n<N/8; n++)
{
    x[2*n] = -yi1[n] * w[2*n] ;
    x[2*n+1] = yr1[N/8-n-1] * w[2*n+1] ;
    x[N/4+2*n] = -yr1[n] * w[N/4+2*n] ;
    x[N/4+2*n+1] = yi1[N/8-n-1] * w[N/4+2*n+1] ;
    x[N/2+2*n] = -yr2[n] * w[N/2-2*n-1] ;
    x[N/2+2*n+1] = yi2[N/8-n-1] * w[N/2-2*n-2] ;
    x[3N/4+2*n] = yi2[n] * w[N/4-2*n-1] ;
    x[3N/4+2*n+1] = -yr2[N/8-n-1] * w[N/4-2*n-2] ;
}
```

where

yr1[n] = real(y1[n]) ;
yi1[n] = imag(y1[n]) ;
yr2[n] = real(y2[n]) ;
yi2[n] = imag(y2[n]) ;

and w[n] is the transform window sequence (see Table 7.33).

6) Overlap and add step.

The first half of the windowed block is overlapped with the second half of the previous block to produce PCM samples (the factor of 2 scaling undoes headroom scaling performed in the encoder):

Pseudo code

```
for(n=0; n<N/2; n++)
{
    pcm[n] = 2 * (x[n] + delay[n]) ;
    delay[n] = x[N/2+n] ;
}
```

Note that the arithmetic processing in the overlap/add processing must use saturation arithmetic to prevent overflow (wrap around). Since the output signal consists of the original signal plus coding error, it is possible for the output signal to exceed 100% level even though the original input signal was less than or equal to 100% level.

Table 7.33 Transform window sequence (w[addr]), where addr = (10 * A) + B.

	B=0	B=1	B=2	B=3	B=4	B=5	B=6	B=7	B=8	B=9
A=0	0.00014	0.00024	0.00037	0.00051	0.00067	0.00086	0.00107	0.00130	0.00157	0.00187
A=1	0.00220	0.00256	0.00297	0.00341	0.00390	0.00443	0.00501	0.00564	0.00632	0.00706
A=2	0.00785	0.00871	0.00962	0.01061	0.01166	0.01279	0.01399	0.01526	0.01662	0.01806
A=3	0.01959	0.02121	0.02292	0.02472	0.02662	0.02863	0.03073	0.03294	0.03527	0.03770
A=4	0.04025	0.04292	0.04571	0.04862	0.05165	0.05481	0.05810	0.06153	0.06508	0.06878
A=5	0.07261	0.07658	0.08069	0.08495	0.08935	0.09389	0.09859	0.10343	0.10842	0.11356
A=6	0.11885	0.12429	0.12988	0.13563	0.14152	0.14757	0.15376	0.16011	0.16661	0.17325
A=7	0.18005	0.18699	0.19407	0.20130	0.20867	0.21618	0.22382	0.23161	0.23952	0.24757
A=8	0.25574	0.26404	0.27246	0.28100	0.28965	0.29841	0.30729	0.31626	0.32533	0.33450
A=9	0.34376	0.35311	0.36253	0.37204	0.38161	0.39126	0.40096	0.41072	0.42054	0.43040
A=10	0.44030	0.45023	0.46020	0.47019	0.48020	0.49022	0.50025	0.51028	0.52031	0.53033
A=11	0.54033	0.55031	0.56026	0.57019	0.58007	0.58991	0.59970	0.60944	0.61912	0.62873
A=12	0.63827	0.64774	0.65713	0.66643	0.67564	0.68476	0.69377	0.70269	0.71150	0.72019
A=13	0.72877	0.73723	0.74557	0.75378	0.76186	0.76981	0.77762	0.78530	0.79283	0.80022
A=14	0.80747	0.81457	0.82151	0.82831	0.83496	0.84145	0.84779	0.85398	0.86001	0.86588
A=15	0.87160	0.87716	0.88257	0.88782	0.89291	0.89785	0.90264	0.90728	0.91176	0.91610
A=16	0.92028	0.92432	0.92822	0.93197	0.93558	0.93906	0.94240	0.94560	0.94867	0.95162
A=17	0.95444	0.95713	0.95971	0.96217	0.96451	0.96674	0.96887	0.97089	0.97281	0.97463
A=18	0.97635	0.97799	0.97953	0.98099	0.98236	0.98366	0.98488	0.98602	0.98710	0.98811
A=19	0.98905	0.98994	0.99076	0.99153	0.99225	0.99291	0.99353	0.99411	0.99464	0.99513
A=20	0.99558	0.99600	0.99639	0.99674	0.99706	0.99736	0.99763	0.99788	0.99811	0.99831
A=21	0.99850	0.99867	0.99882	0.99895	0.99908	0.99919	0.99929	0.99938	0.99946	0.99953
A=22	0.99959	0.99965	0.99969	0.99974	0.99978	0.99981	0.99984	0.99986	0.99988	0.99990
A=23	0.99992	0.99993	0.99994	0.99995	0.99996	0.99997	0.99998	0.99998	0.99998	0.99999
A=24	0.99999	0.99999	0.99999	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
A=25	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000				

7.10 Error Detection

There are several ways in which the AC-3 data may determine that errors are contained within a frame of data. The decoder may be informed of that fact by the transport system which has delivered the data. The data integrity may be checked using the embedded CRCs. Also, some simple consistency checks on the received data can

indicate that errors are present. The decoder strategy when errors are detected is user definable. Possible responses include muting, block repeats, or frame repeats. The amount of error checking performed, and the behavior in the presence of errors are not specified in this standard, but are left to the application and implementation.

7.10.1 CRC Checking

Each AC-3 frame contains two 16-bit CRC words. crc1 is the second 16-bit word of the frame, immediately following the sync word. crc2 is the last 16-bit word of the frame, immediately preceding the sync word of the following frame. crc1 applies to the first 5/8 of the frame, not including the sync word. crc2 provides coverage for the last 3/8 of the frame as well as for the entire frame (not including the sync word). Decoding of CRC word(s) allows errors to be detected.

The following generator polynomial is used to generate each of the 16-bit CRC words: $x^{16} + x^{15} + x^2 + 1$.

The 5/8 of a frame is defined in Table 7.34, and may be calculated by:

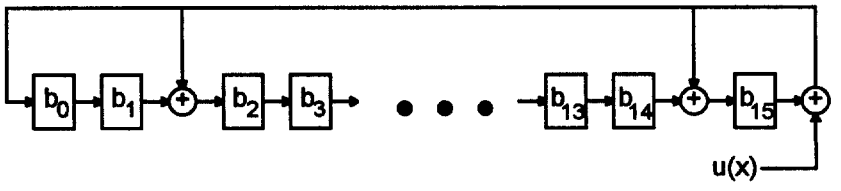
```
5/8_framesize = truncate(framesize ÷ 2) + truncate(framesize ÷ 8) ;
```

or

```
5/8_famesize = (int) (framesize>>1) + (int) (framesize>>3) ;
```

where framesize is in units of 16-bit words. Table 7.34 shows the value of 5/8 of the frame size as a function of AC-3 bit-rate and audio sample rate.

The CRC calculation may be implemented by one of several standard techniques. A convenient hardware implementation is a linear feedback shift register (LFSR). An example of an LFSR circuit for the above generator polynomial is the following:



Checking for valid CRC with the above circuit consists of resetting all registers to zero, and then shifting the AC-3 data bits serially into the circuit in the order in which they appear in the data stream. The sync word is not covered by either CRC (but is included in the indicated 5/8_framesize) so it should not be included in the CRC calculation. crc1 is considered valid if the above register contains all zeros after the first 5/8 of the frame has been shifted in. If the calculation is continued until all data in the frame has been shifted through, and the value is again equal to zero, then crc2 is considered valid. Some decoders may choose to only check crc2, and not check for a valid crc1 at the 5/8 point in the frame. If crc1 is invalid, it is possible to reset the registers to zero and then check crc2. If crc2 then checks, then the last 3/8 of the frame is probably error free. This is of little utility however, since if errors are present in the initial 5/8 of a frame it is not possible to decode any audio from the frame even if the final 3/8 is error free.

Table 7.34 5/8_frame size table; number of words in the first 5/8 of the frame.

frmsizecod	nominal bit-rate	fs = 32 kHz 5/8_framesize	fs = 44.1 kHz 5/8_framesize	fs = 48 kHz 5/8_framesize
000000 (0)	32 kb/s	60	42	40
000001 (0)	32 kb/s	60	43	40
000010 (1)	40 kb/s	75	53	50
000011 (1)	40 kb/s	75	55	50
000100 (2)	48 kb/s	90	65	60
000101 (2)	48 kb/s	90	65	60
000110 (3)	56 kb/s	105	75	70
000111 (3)	56 kb/s	105	76	70
001000 (4)	64 kb/s	120	86	80
001001 (4)	64 kb/s	120	87	80
001010 (5)	80 kb/s	150	108	100
001011 (5)	80 kb/s	150	108	100
001100 (6)	96 kb/s	180	130	120
001101 (6)	96 kb/s	180	130	120
001110 (7)	112 kb/s	210	151	140
001111 (7)	112 kb/s	210	152	140
010000 (8)	128 kb/s	240	173	160
010001 (8)	128 kb/s	240	173	160
010010 (9)	160 kb/s	300	217	200
010011 (9)	160 kb/s	300	217	200
010100 (10)	192 kb/s	360	260	240
010101 (10)	192 kb/s	360	261	240
010110 (11)	224 kb/s	420	303	280
010111 (11)	224 kb/s	420	305	280
011000 (12)	256 kb/s	480	347	320
011001 (12)	256 kb/s	480	348	320
011010 (13)	320 kb/s	600	435	400
011011 (13)	320 kb/s	600	435	400
011100 (14)	384 kb/s	720	521	480
011101 (14)	384 kb/s	720	522	480
011110 (15)	448 kb/s	840	608	560
011111 (15)	448 kb/s	840	610	560
100000 (16)	512 kb/s	960	696	640
100001 (16)	512 kb/s	960	696	640
100010 (17)	576 kb/s	1080	782	720
100011 (17)	576 kb/s	1080	783	720
100100 (18)	640 kb/s	1200	870	800
100101 (18)	640 kb/s	1200	871	800

Note that crc1 is generated by encoders such that the CRC calculation will produce zero at the 5/8 point in the frame. It is *not* the value generated by calculating the CRC of the first 5/8 of the frame using the above generator polynomial. Therefore, decoders should not attempt to save crc1, calculate the CRC for the first 5/8 of the frame, and then compare the two.

Syntactical block size restrictions within each frame (enforced by encoders), guarantee that blocks 0 and 1 are completely covered by crc1. Therefore, decoders may immediately begin processing block 0 when the 5/8 point in the data frame is reached. This may allow smaller input buffers in some applications. Decoders that are able to store an entire frame may choose to process only crc2. These decoders would not begin processing block 0 of a frame until the entire frame is received.

7.10.2 Checking Bit Stream Consistency

It is always possible that an AC-3 frame could have valid sync information and valid CRCs, but otherwise be undecodable. This condition may arise if a frame is corrupted such that the CRC word is nonetheless valid, or in the case of an encoder error (bug). One safeguard against this is to perform some error checking tests within the AC-3 decoder and bit stream parser. Despite its coding efficiency, there are some redundancies inherent in the AC-3 bit stream. If the AC-3 bit stream contains errors, a number of illegal syntactical constructions are likely to arise. Performing checks for these illegal constructs will detect a great many significant error conditions.

The following is a list of known bit stream error conditions. In some implementations it may be important that the decoder be able to benignly deal with these errors. Specifically, decoders may wish to ensure that these errors do not cause reserved memory to be overwritten with invalid data, and do not cause processing delays by looping with illegal loop counts. Invalid audio reproduction may be allowable, so long as system stability is preserved.

1) (blknum == 0) &&
   (cplstre == 0) ;

2) (cplinu == 1) &&
   (no channels in coupling) ;

3) (cplinu == 1) &&
   (cplbegf > (cplendf+2)) ;

4) (cplinu == 1) &&
   ((blknum == 0) || (previous cplinu == 0)) &&
   (chincpl[n] == 1) &&
   (cplcoe[n] == 0) ;

5) (blknum == 0) &&
   (acmod == 2) &&
   (rematstr == 0) ;

6) (cplinu == 1) &&
   ((blknum == 0) || (previous cplinu == 0)) &&
   (cplexpstr == 0) ;

7) (cplinu == 1) &&
   ((cplbegf != previous cplbegf) || (cplendf != previous cplendf)) &&
   (cplexpstr == 0) ;

8) (blknum == 0) &&
   (chexpstr[n] == 0) ;

9) (cplinu == 1) &&
(cplbegf != previous cplbegf) &&
(chincpl[n] == 1) &&
(chexpstr[n] == 0) ;

10) (blknum == 0) &&
(lfeon == 1) &&
(lfeexpstr == 0) ;

11) (chincpl[n] == 0) &&
(chbwcod[n] > 60) ;

12) (blknum == 0) &&
(baie == 0) ;

13) (blknum == 0) &&
(snroffste == 0) ;

14) (blknum == 0) &&
(cplinu == 1) &&
(cplleake == 0) ;

15) (cplinu == 1) &&
(expanded length of cpl delta bit allocation > 50) ;

16) expanded length of delta bit allocation[n] > 50 ;

17) compositely coded 5-level exponent value > 124 ;

18) compositely coded 3-level mantissa value > 26 ;

19) compositely coded 5-level mantissa value > 124 ;

20) compositely coded 11-level mantissa value > 120 ;

21) bitstream unpacking continues past the end of the frame ;

Note that some of these conditions (such as #17 through #20) can only be tested for at low-levels within the decoder software, resulting in a potentially significant MIPS impact. So long as these conditions do not affect system stability, they do not need to be specifically prevented.

8. ENCODING THE AC-3 BIT STREAM

8.1 Introduction

This section provides some guidance on AC-3 encoding. Since AC-3 is specified by the syntax and decoder processing, the encoder is not precisely specified. The only normative requirement on the encoder is that the output elementary bit stream follow AC-3 syntax. Encoders of varying levels of sophistication may be produced. More sophisticated encoders may offer superior audio performance, and may make operation at lower bit-rates acceptable. Encoders are expected to improve over time. All decoders will benefit from encoder improvements. The encoder described in this section, while basic in operation, provides good performance. The description which follows indicates several avenues of potential improvement. A flow diagram of the encoding process is shown in Figure 8.1.

8.2 Summary of the Encoding Process

8.2.1 Input PCM

8.2.1.1 Input Word Length

The AC-3 encoder accepts audio in the form of PCM words. The internal dynamic range of AC-3 allows input wordlengths of up to 24 bits to be useful.

8.2.1.2 Input Sample Rate

The input sample rate must be locked to the output bit rate so that each AC-3 sync frame contains 1536 samples of audio. If the input audio is available in a PCM format at a different sample rate than that required, sample rate conversion must be performed to conform the sample rate.

8.2.1.3 Input Filtering

Individual input channels may be high-pass filtered. Removal of DC components of signals can allow more efficient coding since data rate is not used up encoding DC. However, there is the risk that signals which do not reach 100% PCM level before high-pass filtering will exceed 100% level after filtering, and thus be clipped. A typical encoder would high-pass filter the input signals with a single pole filter at 3 Hz.

The lfe channel should be low-pass filtered at 120 Hz. A typical encoder would filter the lfe channel with an 8th order elliptic filter with a cutoff frequency of 120 Hz.

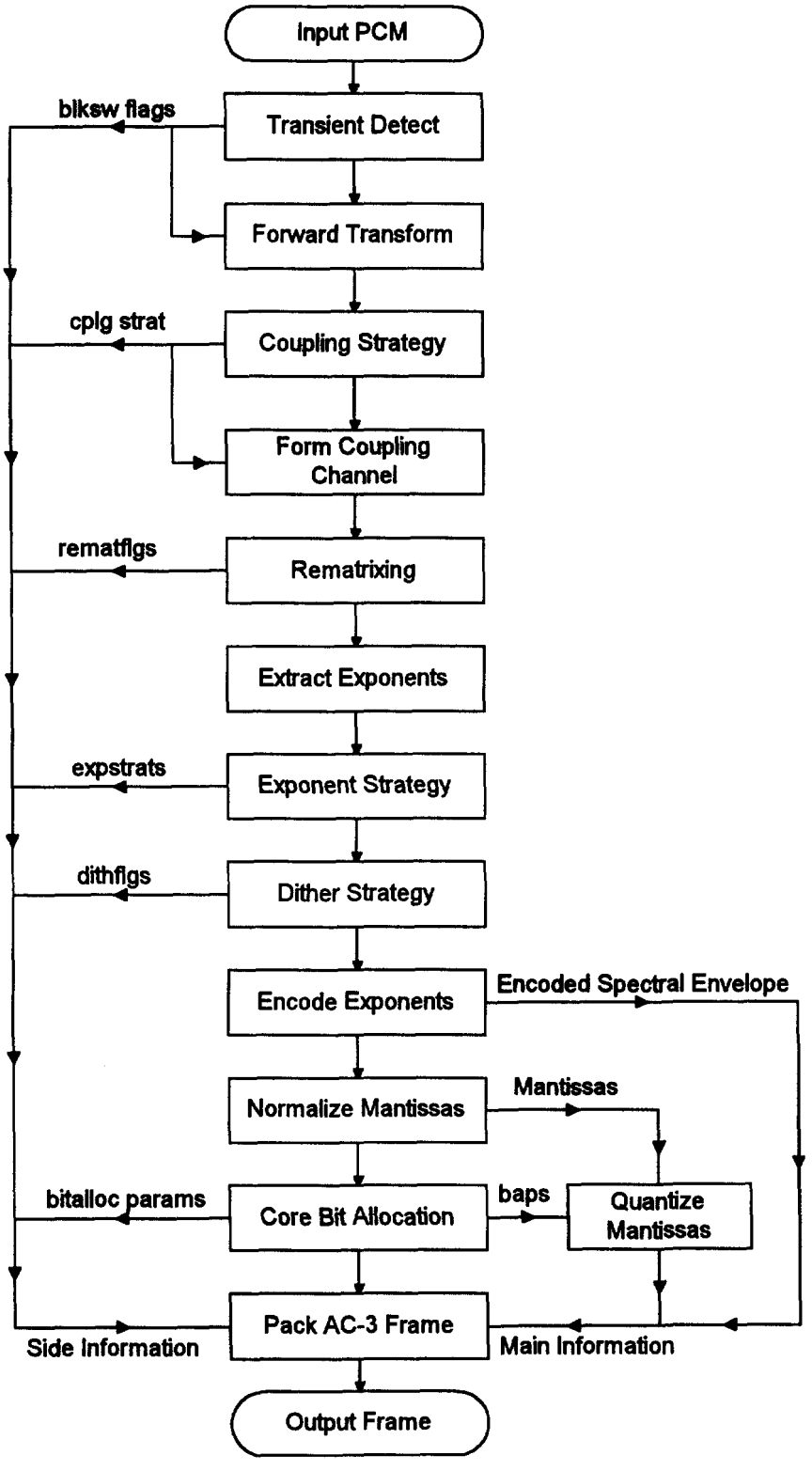


Figure 8.1 Flow diagram of the encoding process.

8.2.2 Transient Detection

Transients are detected in the full-bandwidth channels in order to decide when to switch to short length audio blocks to improve pre-echo performance. High-pass filtered versions of the signals are examined for an increase in energy from one sub-block time-segment to the next. Sub-blocks are examined at different time scales. If a transient is detected in the second half of an audio block in a channel, that channel switches to a short block. A channel that is block-switched uses the D45 exponent strategy.

The transient detector is used to determine when to switch from a long transform block (length 512), to the short block (length 256). It operates on 512 samples for every audio block. This is done in two passes, with each pass processing 256 samples. Transient detection is broken down into four steps: 1) high-pass filtering, 2) segmentation of the block into submultiples, 3) peak amplitude detection within each sub-block segment, and 4) threshold comparison. The transient detector outputs a flag blksw[n] for each full-bandwidth channel, which when set to "one" indicates the presence of a transient in the second half of the 512 length input block for the corresponding channel.

1) High-pass filtering: The high-pass filter is implemented as a cascaded biquad direct form II IIR filter with a cutoff of 8 kHz.

2) Block Segmentation: The block of 256 high-pass filtered samples are segmented into a hierarchical tree of levels in which level 1 represents the 256 length block, level 2 is two segments of length 128, and level 3 is four segments of length 64.

3) Peak Detection: The sample with the largest magnitude is identified for each segment on every level of the hierarchical tree. The peaks for a single level are found as follows:

$$P[j][k] = \max(x(n))$$
$$\text{for } n = (512 \times (k-1) / 2j),\ (512 \times (k-1) / 2j) + 1,\ \ldots (512 \times k / 2j) - 1$$
$$\text{and } k = 1, \ldots 2j-1 ;$$

where: $x(n)$ = the nth sample in the 256 length block
$j = 1, 2, 3$ is the hierarchical level number
k = the segment number within level j

Note that P[j][0], (i.e., k=0) is defined to be the peak of the last segment on level j of the tree calculated immediately prior to the current tree. For example, P[3][4] in the preceding tree is P[3][0] in the current tree.

4) Threshold Comparison: The first stage of the threshold comparator checks to see if there is significant signal level in the current block. This is done by comparing the overall peak value P[1][1] of the current block to a "silence threshold". If P[1][1] is below this threshold then a long block is forced. The silence threshold value is 100/32768. The next stage of the comparator checks the relative peak levels of adjacent segments on each level of the hierarchical tree. If the peak ratio of any two adjacent segments on a particular level exceeds a pre-defined threshold for that level, then a flag is set to indicate the presence of a transient in the current 256 length block. The ratios are compared as follows:

mag(P[j][k]) × T[j] > mag(P[j][(k-1)])
where: T[j] is the pre-defined threshold for level j, defined as:
T[1] = .1
T[2] = .075
T[3] = .05

If this inequality is true for any two segment peaks on any level, then a transient is indicated for the first half of the 512 length input block. The second pass through this process determines the presence of transients in the second half of the 512 length input block.

8.2.3 Forward Transform

8.2.3.1 Windowing

The audio block is multiplied by a window function to reduce transform boundary effects and to improve frequency selectivity in the filter bank. The values of the window function are included in Table 7.33 on page 91. Note that the 256 coefficients given are used back-to-back to form a 512-point symmetrical window.

8.2.3.2 Time to frequency transformation

Based on the block switch flags, each audio block is transformed into the frequency domain by performing one long N=512 point transform, or two short N=256 point transforms. Let x[n] represent the windowed input time sequence. The output frequency sequence, $X_D[k]$ is defined by:

$$X_D[k] = \frac{-2}{N} \sum_{n=0}^{N-1} x[n] \cos\left(\frac{2\pi}{4N}(2n+1)(2k+1) + \frac{\pi}{4}(2k+1)(1+\alpha)\right) \quad \text{for } 0 \le k < N/2$$

where α = -1 for the first short transform
0 for the long transform
+1 for the second short transform

8.2.4 Coupling Strategy

8.2.4.1 Basic encoder

For a basic encoder, a static coupling strategy may be employed. Suitable coupling parameters are:

```
cplbegf = 6 ; /* coupling starts at 10.2 kHz */
cplendf = 12 ; /* coupling channel ends at 20.3 kHz */
cplbndstrc = 1, 0, 1, 1, 0, 1, 1, 1;
cplinu = 1; /* coupling always on */
/* all non-block switched channels are coupled */
for(ch=0; ch<nfchans; ch++) if(blksw[ch]) chincpl[ch] = 1; else chincpl[ch] = 0;
```

Coupling coordinates for all channels may be transmitted for every other block, i.e. blocks 0, 2, and 4. During blocks 1, 3, and 5, coupling coordinates are reused.

8.2.4.2 Advanced Encoder

More advanced encoders may make use of dynamically variable coupling parameters. The coupling frequencies may be made variable based on bit demand and on a psychoacoustic model which compares the audibility of artifacts caused by bit starvation vs those caused by the coupling process. Channels with a rapidly time varying power level may be removed from coupling. Channels with slowly varying power levels may have their coupling coordinates sent less often. The coupling band structure may be made dynamic.

8.2.5 Form Coupling Channel

8.2.5.1 Coupling Channel

The most basic encoder can form the coupling channel by simply adding all of the individual channel coefficients together, and dividing by 8. The division by 8 prevents the coupling channel from exceeding a value of 1. Slightly more sophisticated encoders can alter the sign of individual channels before adding them into the sum so as to avoid phase cancellations.

8.2.5.2 Coupling Coordinates

Coupling coordinates are formed by taking power ratios within of each coupling band. The power in the original channel within a coupling band is divided by the power in the coupling channel within the coupling band. This power ratio becomes the coupling coordinate. The coupling coordinates are converted to floating point format and quantized. The exponents for each channel are examined to see if they can be further scaled by 3, 6, or 9. This generates the 2-bit master coupling coordinate for that channel. (The master coupling coordinates allow the dynamic range represented by the coupling coordinate to be increased.)

8.2.6 Rematrixing

Rematrixing is active only in the 2/0 mode. Within each rematrixing band, power measurements are made on the L, R, L+R, and L-R signals. If the maximum power is found in the L or R channels, the rematrix flag is not set for that band. If the maximum power is found in the L+R or L-R signal, then the rematrix flag is set. When the rematrix flag for a band is set, the encoder codes L+R and L-R instead of L and R. Rematrixing is described in Section 7.5 on page 70.

8.2.7 Extract Exponents

The binary representation of each frequency coefficient is examined to determine the number of leading zeros. The number of leading zeroes (up to a maximum of 24) becomes the initial exponent value. These exponents are extracted and the exponent sets

(one for each block for each channel, including the coupling channel) are used to determine the appropriate exponent strategies.

8.2.8 Exponent Strategy

For each channel, the variation in exponents over frequency and time is examined. If the exponents indicate a relatively flat spectrum, an exponent strategy such as D25 or D45 may be used. If the spectrum is very tonal, then a high spectral resolution exponent strategy such as D15 or D25 would be used. If the spectrum changes little over the 6 blocks in a frame, the exponents may be sent only for block 0, and reused for blocks 1-5. If the exponents are changing rapidly during the frame, exponents may be sent for block 0 and for those blocks which have exponent sets which differ significantly from the previously sent exponents. There is a tradeoff between fine frequency resolution, fine time resolution, and the number of bits required to send exponents. In general, when operating at very low bit rates, it is necessary to trade off time vs frequency resolution.

In a basic encoder a simple algorithm may be employed. First, look at the variation of exponents over time. When the variation exceeds a threshold new exponents will be sent. The exponent strategy used is made dependent on how many blocks the new exponent set is used for. If the exponents will be used for only a single block, then use strategy D45. If the new exponents will be used for 2 or 3 blocks, then use strategy D25. If the new exponents will be used for 4,5, or 6 blocks, use strategy D15.

8.2.9 Dither Strategy

The encoder controls, on a per channel basis, whether coefficients which will be quantized to zero bits will be reproduced with dither. The intent is to maintain approximately the same energy in the reproduced spectrum even if no bits are allocated to portions of the spectrum. Depending on the exponent strategy, and the accuracy of the encoded exponents, it may be beneficial to defeat dither for some blocks.

A basic encoder can implement a simple dither strategy on a per channel basis. When blksw[ch] is 1, defeat dither for that block and for the following block.

8.2.10 Encode Exponents

Based on the selected exponent strategy, the exponents of each exponent set are preprocessed. D25 and D45 exponent strategies require that a single exponent be shared over more than one mantissa. The exponents will be differentially encoded for transmission in the bit stream. The difference between successive raw exponents does not necessarily produce legal differential codes (maximum value of ±2) if the slew rate of the raw exponents is greater than that allowed by the exponent strategy. Preprocessing adjusts exponents so that transform coefficients that share an exponent have the same exponent and so that differentials are legal values. The result of this processing is that some exponents will have their values decreased, and the corresponding mantissas will have some leading zeroes.

The exponents are differentially encoded to generate the encoded spectral envelope. As part of the encoder processing, a set of exponents is generated which is equal to the set of exponents which the decoder will have when it decodes the encoded spectral envelope.

8.2.11 Normalize Mantissas

Each channel's transform coefficients are normalized by left shifting each coefficient the number of times given by its corresponding exponent to create normalized mantissas. The original binary frequency coefficients are left shifted according to the exponents which the decoder will use. Some of the normalized mantissas will have leading zeroes. The normalized mantissas are what are quantized.

8.2.12 Core Bit Allocation

A basic encoder may use the core bit allocation routine with all parameters fixed at nominal default values.

```
sdcycod = 2 ;
fdcycod = 1 ;
sgaincod = 1 ;
dbpbcod = 2 ;
floorcod = 4 ;
cplfgaincod = 4 ;
fgaincod[ch] = 4 ;
lfegaincod = 4 ;
cplsnroffst = fsnroffst[ch] = lfesnroffst = fineoffset ;
```

Since the bit allocation parameters are static, they are only sent during block 0. Delta bit allocation is not used, so deltbaie = 0. The core bit allocation routine (described in Section 7.2 on page 49) is run, and the coarse and fine SNR offsets are adjusted until all available bits in the frame are used up. The coarse SNR offset adjusts in 6 dB increments, and the fine offset adjusts in 3/8 dB increments. Bits are allocated globally from a common bit pool to all channels. The combination of csnroffst and fineoffset are chosen which uses the largest number of bits without exceeding the frame size. This involves an iterative process. When, for a given iteration, the number of bits exceeds the pool, the SNR offset is decreased for the next iteration. On the other hand, if the allocation is less than the pool, the SNR offset is increased for the next iteration. When the SNR offset is at its maximum without causing the allocation to exceed the pool, the iterating is complete. The result of the bit allocation routine are the final values of csnroffst and fineoffest, and the set of bit allocation pointers (baps). The SNR offset values are included in the bit stream so that the decoder does not need to iterate.

8.2.13 Quantize Mantissas

The baps are used by the mantissa quantization block. There is a bap for each individual transform coefficient. Each normalized mantissas is quantized by the quantizer indicated by the corresponding bap. Asymmetrically quantized mantissas are quantized by rounding to the number of bits indicated by the corresponding bap. Symmetrically

quantized mantissas are quantized through the use of a table lookup. Mantissas with baps of 1, 2, and 4 are grouped into triples or duples.

8.2.14 Pack AC-3 Frame

All of the data is packed into the encoded AC-3 frame. Some of the quantized mantissas are grouped together and coded by a single codeword. The output format is dependent on the application. The frame may be output in a burst, or delivered as a serial data stream at a constant rate.

Blank Page

ANNEX A

(Normative)

AC-3 ELEMENTARY STREAMS IN AN MPEG-2 MULTIPLEX

1. SCOPE

This Annex contains specifications on how to combine one or more AC-3 elementary streams into an MPEG-2 "Transport Stream" or "Program Stream" (ISO/IEC 13818-1). Applications which reference this specification may need to specify precisely the values of some of the parameters described in this Annex.

2. INTRODUCTION

The AC-3 elementary bit stream is included in an MPEG-2 multiplex bit stream in much the same way an MPEG-1 audio stream would be included. The AC-3 bit stream is packetized into PES packets. An MPEG-2 multiplex bit stream containing AC-3 elementary streams must meet all audio constraints described in the STD model in Section 3.6. It is necessary to unambiguously indicate that an AC-3 stream is, in fact, an AC-3 stream (and not an MPEG audio stream). The MPEG-2 standard does not explicitly indicate codes to be used to indicate an AC-3 stream. Also, the MPEG-2 standard does not have an audio descriptor adequate to describe the contents of the AC-3 bit stream in the PSI tables.

The AC-3 audio access unit (AU) or presentation unit (PU) is an AC-3 sync frame. The AC-3 sync frame contains 1536 audio samples. The duration of an AC-3 access (or presentation) unit is 32 ms for audio sampled at 48 kHz, approximately 34.83 ms for audio sampled at 44.1 kHz, and 48 ms for audio sampled at 32 kHz.

The items which need to be specified in order to include AC-3 within the MPEG-2 bit stream are: stream_type, stream_id, registration descriptor, and AC-3 audio descriptor. The use of the ISO 639 language descriptor is optional. Some constraints are placed on the PES layer for the case of multiple audio streams intended to be reproduced in exact sample synchronism.

3. DETAILED SPECIFICATION

3.1 Stream_type

The preferred value of stream_type for AC-3 is 0x81. Other values which MPEG has assigned as "User Private" may be used also. If the value of 0x81 is used, depending on the particular application, the decoder may assume that stream_type 0x81 indicates AC-3 audio. If the potential for ambiguity exists, the AC-3 registration descriptor should be included (see Annex A, Section 3.3).

3.2 Stream_id

3.2.1 Transport stream

For transport streams, the value of stream_id in the PES header shall be 0xBD (indicating private_stream_1). Multiple AC-3 streams may share the same value of stream_id since each stream is carried with a unique PID value. The mapping of values of PID to stream_type is indicated in the transport stream Program Map Table (PMT).

3.2.2 Program stream

In program streams, the stream_id is intended to specify the type and number of the elementary stream. Multiple AC-3 elementary streams can not use a common value of stream_id; unique values are required. If a single AC-3 elementary stream is carried in a program stream, stream_id may use the value 0xBD (indicating private_stream_1). ISO/IEC 13818-1 does not provide values of stream_id suitable for identifying multiple AC-3 elementary streams. If multiple AC-3 elementary streams are carried in a program stream, stream_id shall use the values 110x xxxx, where x xxxx indicates a stream number with a value of 0-31. This value for stream_id is identical to the value used for MPEG-1 or MPEG-2 audio. Confusion between MPEG audio and AC-3 audio may be avoided by a Program Stream Map, which associates values of stream_id with values of stream_type. Streams which use a stream_id of 110x xxxx are clearly identified as to the type of audio coding employed by the value of stream_type which is linked to the each value of stream_id.

3.3 Registration descriptor

The syntax of the AC-3 registration descriptor is shown in Table 1. If the stream_type value used for AC-3 is not 0x81, then the AC-3 registration descriptor shall be included in the TS_program_map_section (for transport streams) or the program_stream_map (for program streams). If the stream_type value used for AC-3 is 0x81, the AC-3 registration may be included optionally (it should be included if there is any chance of ambiguity).

Table 1 AC-3 Registration Descriptor

Syntax	No. of bits	Mnemonic
registration_descriptor() {		
descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
format_identifier	**32**	**uimsbf**
}		

descriptor_tag — 0x05.

descriptor_length — 0x04.

format_identifier — The AC-3 format_identifier is 0x41432D33 ("AC-3").

3.4 AC-3 audio descriptor

The AC-3 audio descriptor, shown in Table 2, allows information about individual AC-3 elementary streams to be included in the program specific information (PSI) tables. This information is useful to allow the appropriate AC-3 stream(s) to be directed to the audio decoder. Note that horizontal lines in the table indicate allowable termination points for the descriptor.

Table 2 AC-3 Audio Descriptor Syntax

Syntax	No. of bits	Mnemonic
audio_stream_descriptor( ) {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
sample_rate_code	3	bslbf
bsid	5	bslbf
bit_rate_code	6	bslbf
surround_mode	2	bslbf
bsmod	3	bslbf
num_channels	4	bslbf
full_svc	1	bslbf
langcod	8	bslbf
if(num_channels==0) /* 1+1 mode */		
langcod2	8	bslbf
if(bsmod<2) {		
mainid	3	uimsbf
reserved	5	bslbf
}		
else asvcflags	8	bslbf
textlen	7	uimsbf
text_code	1	bslbf
for(i=0; i<M; i++) {		
text[i]	8	bslbf
}		
for(i=0; i<N; i++) {		
additional_info[i]	N×8	bslbf
}		
}		

descriptor_tag — The preferred value for the AC-3 descriptor tag is 0x81. Other values which MPEG has assigned as User Private may also be used.

descriptor_length — This is an 8-bit field specifying the number of bytes of the descriptor immediately following descriptor_length field.

sample_rate_code — This is a 3-bit field which indicates the sample rate of the encoded audio. The indication may be of one specific sample rate, or may be of a set of values which include the sample rate of the encoded audio. See Table 3.

Table 3 Sample Rate Code Table

sample_rate_code	Sample rate
'000'	48 kHz
'001'	44.1 kHz
'010'	32 kHz
'011'	reserved
'100'	48 kHz or 44.1 kHz
'101'	48 kHz or 32 kHz
'110'	44.1 kHz or 32 kHz
'111'	48 kHz or 44.1 kHz or 32 kHz

bsid — This is a 5-bit field which is set to the same value as the bsid field in the AC-3 elementary stream.

bit_rate_code — This is a 6-bit field. The lower 5 bits indicate a nominal bit rate. The MSB indicates whether the indicated bit rate is exact (MSB=0) or an upper limit (MSB=1). See Table 4.

Table 4 Bit Rate Code Table

bit_rate_code	exact bit rate	bit_rate_code	bit rate upper limit
'000000' (0.)	32 kbps	'100000' (32.)	32 kbps
'000001' (1.)	40 kbps	'100001' (33.)	40 kbps
'000010' (2.)	48 kbps	'100010' (34.)	48 kbps
'000011' (3.)	56 kbps	'100011' (35.)	56 kbps
'000100' (4.)	64 kbps	'100100' (36.)	64 kbps
'000101' (5.)	80 kbps	'100101' (37.)	80 kbps
'000110' (6.)	96 kbps	'100110' (38.)	96 kbps
'000111' (7.)	112 kbps	'100111' (39.)	112 kbps
'001000' (8.)	128 kbps	'101000' (40.)	128 kbps
'001001' (9.)	160 kbps	'101001' (41.)	160 kbps
'001010' (10.)	192 kbps	'101010' (42.)	192 kbps
'001011' (11.)	224 kbps	'101011' (43.)	224 kbps
'001100' (12.)	256 kbps	'101100' (44.)	256 kbps
'001101' (13.)	320 kbps	'101101' (45.)	320 kbps
'001110' (14.)	384 kbps	'101110' (46.)	384 kbps
'001111' (15.)	448 kbps	'101111' (47.)	448 kbps
'010000' (16.)	512 kbps	'110000' (48.)	512 kbps
'010001' (17.)	576 kbps	'110001' (49.)	576 kbps
'010010' (18.)	640 kbps	'110010' (50.)	640 kbps

dsurmod — This is a 2-bit field which may be set to the same value as the dsurmod field in the AC-3 elementary stream, or which may be set to '00' (not indicated). See Table 5.

Table 5 Dsurmod Table

surround_mode	Meaning
'00'	not indicated
'01'	NOT Dolby Surround encoded
'10'	Dolby Surround encoded
'11'	reserved

bsmod — This is a 3-bit field which is set to the same value as the bsmod field in the AC-3 elementary stream.

num_channels — This is a 4-bit field which indicates the number of channels in the AC-3 elementary stream. When the MSB is 0, the lower 3 bits are set to the same value as the acmod field in the AC-3 elementary stream. When the MSB field is 1, the lower 3 bits indicate the maximum number of encoded audio channels (counting the LFE channel as 1). If the value of acmod in the AC-3 elementary stream is '000' (1+1 mode), then the value of num_channels shall be set to '0000'. See Table 6.

Table 6 Num_Channels Table

num_channels	audio coding mode (acmod)	num_channels	number of encoded channels
'0000'	1+1	'1000'	1
'0001'	1/0	'1001'	≤ 2
'0010'	2/0	'1010'	≤ 3
'0011'	3/0	'1011'	≤ 4
'0100'	2/1	'1100'	≤ 5
'0101'	3/1	'1101'	≤ 6
'0110'	2/2	'1110'	reserved
'0111'	3/2	'1111'	reserved

full_svc — This is a 1-bit field which indicates whether or not this audio service is a full service suitable for presentation, or whether this audio service is only a partial service which should be combined with another audio service before presentation. This bit should be set to a '1' if this audio service is sufficiently complete to be presented to the listener without being combined with another audio service (for example, a visually impaired service which contains all elements of the program; music, effects, dialogue, and the visual content descriptive narrative). This bit should be set to a '0' if the service is not sufficiently complete to be presented without being combined with another audio service (e.g., a visually impaired service which only contains a narrative description of the visual program content and which needs to be combined with another audio service which contains music, effects, and dialogue).

langcod — This is an 8-bit field which is set to the same value as the langcod field in the AC-3 elementary stream. A value of 0x00 indicates that the language is unknown or not indicated.

langcod2 — This is an 8-bit field which is set to the value of the langcod2 field in the AC-3 elementary stream. This field indicates the language of the audio contained in the second mono audio channel (1+1 mode only).

mainid — This is a 3-bit field which contains a number in the range 0-7 which identifies a main audio service. Each main service should be tagged with a unique number. This value is used as an identifier to link associated services with particular main services.

asvcflags — This is an 8-bit field. Each bit (0-7) indicates with which main service(s) this associated service is associated. The left most bit, bit 7, indicates whether this associated service may be reproduced along with main service number 7. If the bit has a value of 1, the service is associated with main service number 7. If the bit has a value of 0, the service is not associated with main service number 7.

textlen — This is an unsigned integer which indicates the length, in bytes, of a descriptive text field which follows.

text_code — This is a 1-bit field which indicates how the following text field is encoded. If this bit is a '1', the text is encoded as 1-byte characters using the ISO Latin-1 alphabet (ISO 8859-1). If this bit is a '0', the text is encoded with 2-byte unicode characters.

text[i] — The text field may contain a brief textual description of the audio service.

additional_info[i] — This is a set of additional bytes filling out the remainder of the descriptor. The purpose of these bytes is not currently defined. This field is provided to allow the descriptor to be extended in the future.

3.5 ISO_639_language_code

The ISO_639_language_code descriptor allows a stream to be tagged with the 24-bit ISO 639 language code. The AC-3 bit stream and the AC-3 audio descriptor both contain an (identical) 8-bit language code which is adequate for most applications. Additional use of the ISO_639_language_code descriptor is thus redundant. If the ISO_639_language_descriptor is included in the TS_program_map_section (for transport streams) or the program_stream_map (for program streams), then the audio_type field of this descriptor shall have a value of 0x00 (undefined).

3.6 STD audio buffer size

For an MPEG-2 transport stream, the T-STD model defines the main audio buffer size (BS_n) as:

$$BS_n = BS_{mux} + BS_{dec} + BS_{oh}$$

where

BS_{mux} = 736 bytes

BS_{oh} = PES header overhead

BS_{dec} = access unit buffer.

MPEG-2 specifies a fixed value for BS_n (3584 bytes) and indicates that any excess buffer may be used for additional multiplexing.

When an AC-3 elementary stream is carried by an MPEG-2 transport stream, the transport stream shall be compliant with a main audio buffer size of:

$$BS_n = BS_{mux} + BS_{pad} + BS_{dec}$$

where

BS_{mux} = 736 bytes

BS_{pad} = 64 bytes

BS_{dec} may be found in Table 5.13 of ATSC Standard A/52 (for the case of 44.1 kHz sample rate, the larger of the two values shown shall be used). The 64 bytes in BS_{pad} are available for BS_{oh} and additional multiplexing. This constraint makes it possible to implement decoders with the minimum possible memory buffer.

Applications which employ program streams should specify appropriate constraints.

4. PES CONSTRAINTS

4.1 Encoding

In some applications, the audio decoder may be capable of simultaneously decoding two elementary streams containing different program elements, and then combining the program elements into a complete program. Most of the program elements are found in the *main audio service*. Another program element (such as a narration of the picture content intended for the visually impaired listener) may be found in the *associated audio service*. In this case the audio decoder may sequentially decode audio frames (or audio blocks) from each elementary stream and do the combining (mixing together) on a frame or (block) basis. In order to have the audio from the two elementary streams reproduced in exact sample synchronism, it is necessary for the original audio elementary stream encoders to have encoded the two audio program elements frame synchronously; i.e., if audio stream 1 has sample 0 of frame n taken at time t0, then audio stream 2 should also have frame n beginning with its sample 0 taken the identical time t0. If the encoding of multiple audio services is done frame and sample synchronous, and decoding is intended to be frame and sample synchronous, then the PES packets of these audio services shall contain identical values of PTS which refer to the audio access units intended for synchronous decoding.

Audio services intended to be combined together for reproduction shall be encoded at an identical sample rate.

4.2 Decoding

If audio access units from two audio services which are to be simultaneously decoded have identical values of PTS indicated in their corresponding PES headers, then

the corresponding audio access units shall be presented to the audio decoder for simultaneous synchronous decoding. Synchronous decoding means that for corresponding audio frames (access units), corresponding audio samples are presented at the identical time.

If the PTS values do not match (indicating that the audio encoding was not frame synchronous) then the audio frames (access units) of the main audio service shall be presented to the audio decoder for decoding and presentation at the time indicated by PTS. An associated service which is being simultaneously decoded should have its audio frames (access units), which are in closest time alignment (as indicated by PTS) to those of the main service being decoded, presented to the audio decoder for simultaneous decoding. In this case the associated service may be reproduced out of sync by as much as 1/2 of a frame time. (This is typically satisfactory; a visually impaired narration does not require highly precise timing.)

4.3 Byte-alignment

The AC-3 elementary stream shall be byte-aligned within the MPEG-2 data stream. This means that the initial 8 bits of an AC-3 frame shall reside in a single byte which is carried by the MPEG-2 data stream.